AI on Classic Video Games Using Reinforcement Learning

By
Shivika Sodhi

Project Advisor:
Dr. Chris Pollett

Committee Members:
Dr Jenny Lam
Dr. Robert Chun
Outline

• Introduction
• Archon
• Related Work
• Reinforcement Learning
  • Q-learning Algorithm
• Deep Learning
  • Convolutional Neural Network Algorithm
• Approach 1
• Approach 2
• Conclusion, Result, Future Work, References
Introduction

Project Goals

Primary Goal:

• Design an artificially intelligent bot to play a classic video game, Archon: the light and dark.

• The agent learns on its own to automatically determine successful strategies that lead to the greatest long-term rewards.
Related Work

TD Gammon

• Computer backgammon program

• Developed in 1992 by Gerald Tesauro at IBM

• Achieved a level of play just slightly below that of the top human backgammon players of the time
Related Work

Playing Atari with Deep Reinforcement Learning

• Remarkable results by DeepMind
• Demonstrated how a computer learned to play Atari 2600 video games by observing just the screen pixels and receiving a reward when the game score increased
• Gave state-of-the-art results in six of the seven games it was tested on
• No adjustment of the architecture or hyperparameters.
Archon

The Light and the Dark

• Classic video game, developed in 1983 by Free Fall Associates

• Similar to chess, apart from the fight mode

• Generally in combat, a stronger player defeats a weaker player in either defending or capturing a square
Libraries used

For image processing

• OpenCV-Python
  • aimed at real-time computer vision

• Template matching
  • technique for finding areas of an image that match (are similar) to a template image (patch).
  • **Source image**: image in which we expect to find a match to the template image
  • **Template image**: patch image that will be compared to the template image
Libraries used

For automating the bot

- PyautoGUI
- Keyboard
- PIL (Python Imaging Library)
Introduction

Reinforcement Learning

• General purpose framework for decision-making
• Inspired by behaviorist psychology
• Between supervised and unsupervised learning
• Agent learns to interact with an environment through occasional feedback
• The reward, to achieve a long-term goal.
• Each action by the agent influences its future state
How Dog Training Works

1. Before Conditioning
   - Unconditioned Stimulus: Food
   - Unconditioned Response: Salivation
   - Response: Food

2. Before Conditioning
   - Neutral Stimulus: Bell
   - No Salivation

3. During Conditioning
   - Unconditioned Stimulus: Bell + Food
   - Unconditioned Response: Salivation

4. After Conditioning
   - Conditioned Stimulus: Bell
   - Conditioned Response: Salivation
Markov Decision Process

• Framework for modeling decision making situations
• Outcomes are partly random and partly under the control of an agent
• Set of states and actions, together with rules for transitioning from one state to another, make up a Markov decision process
• RL problem represented as a Markov Decision Process
Credit Assignment Problem

The problem?

- Agent performs an action and receives a positive reward
- No relation with actions performed just before getting that reward
- Q-learning propagates rewards back in time, until it reaches the crucial decision point which was the actual cause for the obtained reward.

Was action “a” the only responsible action for getting the reward R?
Discounted Future Reward

- To perform well in the long-term, we need to take into account not only the immediate rewards, but also the future rewards.
- Our environment is stochastic.
- The more we dive into the future, the more it may diverge.
- Discounted future reward:

\[ R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \ldots + \gamma^{n-t} r_n \]
Exploration-Exploitation

The problem?

- An action plan to obtain certain rewards has been figured out.

- Is it feasible to keep following the same action plan or experiment with something new that could result in a larger reward?
Exploration-Exploitation

$\epsilon$-greedy exploration

- Agent takes the estimated optimal action (with the highest Q-value) most of the time and a random action with probability $\epsilon$

- This ensures that the bot explores the search space and sees how actions not currently considered optimal would have fared instead
Components of RL

- **Policy**
  - Rule how we choose an action in each state
  - Maps from state to action ($a = \pi(s)$)

- **Value Function**
  - $Q(s_t, a_t) = \max R_{t+1}$
  - Best possible score at the end of the game after performing action $a$ in state $s$
  - Prediction of future reward
  - Evaluate state and select between actions

- **Model**
  - Agent’s representation of the environment

\[
\pi(s) = \arg\max_a Q(s, a)
\]
Q learning

• Model-free reinforcement learning technique
• Consists of an agent, states S and a set of actions per state A
• Agent moves from s -> s’ by performing \( a \in A \)
• Executing an action in a specific state provides the agent with a reward
• Goal of the agent is to maximize its total reward (by learning the optimal action for each state)
• Optimal action has highest-long term reward \( R \)
• \( R \): weighted sum of the expected values of all future steps staring from the current state
Q-learning Algorithm

1. Initialize Q
2. Choose an action from Q
3. Perform Action
4. Measure Reward
5. Update Q
Without any form of training
Approach 1
This action has the highest reward, hence action A1 will be chosen.
Initialize length of memory D

Initialize dictionary $Q_{sa}$ with random weights for each action

For each episode of the game

Initialize sequence $s$

While not goal

Select an action based on epsilon-greedy

Execute action in emulator, take a screenshot, calculate reward

Update Sequence and $D$

Sample random minibatch transitions from $D$

Update reward in dictionary $Q_{sa}$

End
Approach 2
Deep Neural Network

- Uses a cascade of many layers of nonlinear processing units for feature extraction and transformation
- Input of the algorithm is raw pixels
- Works using minimal domain knowledge
Convolutional Neural Network

- Act similarly to human receptive fields.
- Make sense of the game’s screen output in a way that is similar to how humans are able to
- Several layers of convolutions with nonlinear activation functions applied to the results
CNN

- Sliding window function applied to a matrix
- Instead of considering each pixel independently, convolutional layers allow us to consider regions of an image
- Build edges from pixels, shapes from edges, and more complex objects from shapes.
OUR DEEP LEARNING MODEL

Input Image

convolutional neural net

Possible Actions:
- Up
- Down
- Left
- Right
- Fire
- Stay
Network architecture

Convolution

Input Layer
- 8 x 8
- 4 frames
- 16 filters

1st hidden
- 4 x 4
- 20 x 20

2nd hidden
- 9 x 9

Fully connected
- 3rd hidden (256 fully connected)
- Output (actions)
Network Architecture

**Input**
- 4 Image of size 80x80 stacked together

**CNN Model (Keras)**
- Conv2D layer
  - 16 filters
  - Of size 8x8 with stride 4

**Output**
- Dense Layer 6 neurons
- Dense Layer 256 neurons
- Conv2D layer
  - 32 filters
  - Of size 4x4 with stride 2
Initialize length of memory D

Initialize Q with random weights for each action

For each episode of the game

Initialize sequence s

While not goal

Select an action based on epsilon-greedy

Execute action in emulator, take a screenshot, calculate reward

Update Sequence and D

Sample random minibatch transitions from D

Compute:

\[ Q(s_t, a_t) = r_t \quad \text{(if terminal)} \]

\[ Q(s_t, a_t) = r_t + \gamma \max Q(s_{t+1}, a_{t+1}) \]

Train neural network model based on loss

End
ARCHON
MADE WITH CAMTIA
FREE FALL GAME
Result

- Experiments were performed on 50 iterations of the game, for two different policies.

- **First approach:**
  - Agent starts playing better gradually, but its improvement is very slow.
  - Survives the fight for approximately 5 more seconds.

- **Second approach:**
  - Agent learns comparatively faster with neural network as compared to when trained without neural networks.
  - Survives the game for about 8 more seconds on the 50th iteration of the game.
Performance of the agent (without Neural Nets)

Length of the fight mode (seconds)

Number of fight modes
Conclusion

• This research demonstrated that reinforcement learning, when implemented with neural networks, perform a better and faster training of the bot as compared to when the Q function is used as a table.

• We trained an artificially intelligent bot to play Archon using two different policies
# System Configuration

<table>
<thead>
<tr>
<th>Windows edition</th>
<th>Windows 10 Home</th>
</tr>
</thead>
<tbody>
<tr>
<td>© 2016 Microsoft Corporation. All rights reserved.</td>
<td>Windows 10</td>
</tr>
</tbody>
</table>

## System

<table>
<thead>
<tr>
<th>Processor:</th>
<th>Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installed memory (RAM):</td>
<td>16.0 GB</td>
</tr>
<tr>
<td>System type:</td>
<td>64-bit Operating System, x64-based processor</td>
</tr>
</tbody>
</table>
Future Scope

• Train using a GPU and thus compare performance
• Train the bot to fight players other than the Knight (light side) and Golem (dark side)
• Train the bot to play the game from the dark side
• Apply the same technique to the strategy mode of the game and finish the game
References

- https://www.slideshare.net/nikolaypavlov/deep-qlearning
Thank you.
Commodore 64

- 8-bit home computer introduced in January 1982 by Commodore International
- Emulator from [http://www.ccs64.com](http://www.ccs64.com)
- Low resolution, less pixels, easier for image processing techniques
Appendix

Q-learning Pseudocode

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize $s$
  Repeat (for each step of episode):
    Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Take action $a$, observe $r, s'$
    $Q(s, a) \leftarrow Q(s, a) + \alpha \left[ r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$
    $s \leftarrow s'$
  until $s$ is terminal
Pseudocode

Algorithm 1 Q function as a table

1: procedure QLEARN
2: Initialize replay memory D to capacity N.
3: Initialize Q with random weights for each action
4: for each episode of game do
5:  seq ← [sₜ, aₜ, sₜ, aₜ, sₜ]
6:  while not goal do
7:     Based on the value of epsilon
8:     either select a random action
9:     or select action with maximum weight
10:    Execute action in emulator
11:    Take a screenshot
12:    Calculate the reward
13:    Calculate the state
14:    Update seq ← [sₜ₊₁, aₜ₊₂, sₜ₊₂, aₜ₊₃, sₜ₊₃]
15:    D ← [sₜ, aₜ, rₜ, sₜ₊₁]
16:    Sample random minibatch transitions from D
17:    Compute: \( Q(sₜ, aₜ) = rₜ + \gamma \max Q(sₜ₊₁, aₜ₊₁) \)
18:    \( sₜ₊₁ \leftarrow sₜ \)
19:  End while
20: End for
Algorithm 2 Deep Q learning algorithm

1: procedure QLEARN
2: Initialize replay memory D to capacity N.
3: Initialize Q with random weights for each action
4: for each episode of game do
5: \( s_t \leftarrow \text{image} \)
6: \( \text{seq} \leftarrow [s_t, s_t, s_t, s_t] \)
7: while not goal do
8: Based on the value of epsilon
9: either select a random action
10: or select action with maximum weight
11: Execute action in emulator
12: Take a screenshot
13: Calculate the reward
14: Calculate the state
15: Update seq \( \leftarrow [s_t, s_t, s_t, s_t, s_{t+1}] \)
16: D \( \leftarrow [s_t, a_t, r_t, s_{t+1}] \)
17: Sample random minibatch transitions from D
18: Predict Q values from the neural network model
19: Compute: \( Q(s_t, a_t) = r_t + \gamma \max Q(s_{t+1}, a_{t+1}) \)
20: \( s_{t+1} \leftarrow s_t \)
21: Calculate loss to update weights
22: End while
23: End for
Training of neural networks
Figure 3: Left: Naive formulation of deep Q-network. Right: More optimized architecture of deep Q-network, used in DeepMind paper.